Towards software architecture runtime models for continuous adaptive monitoring

Thomas Brand, Holger Giese

14.10.2018
Agenda

- **Show why it is relevant** to investigate and support:
  - Continuous adaptive monitoring
  - Modeling languages for long living runtime model instances
- **Demonstrate the significance** of the modeling language
- **Describe the planned roadmap** for proposing an evaluated solution
- **Derive requirements** from illustrative scenarios and indicate how they are supported by two existing approaches
- **Questions and discussion**
Why does monitoring need to be adaptive?

Why is monitoring adaptation without interruption important?

How does the runtime model modeling language relate to this?
“models@run.time is an **abstraction** of a **running system** that is being **manipulated at runtime** for a **specific purpose**”

[Bencomo.2013]

Please imagine a software architecture runtime model thinking of:

- **graph in a datastore**
- **running system**
- **current monitoring results**
- **analysis and phenomena detection processes**
Classical Model-Driven Engineering approach

QueryService

pooledConnectionsCount: Integer
pooledConnectionsMax: Integer

Meta-model level
Runtime model level
System representation content
Modeling language definition content

Modeling language implementation with an API to create models

1.) generate

Query based on types e.g.:
QueryService.
pooledConnectionCount > 5

2.) use

3.) instantiate

4.) access

QueryService

pooledConnectionsCount: Integer
pooledConnectionsMax: Integer

<<instanceOf>>

qs1: QueryService

pooledConnectionsCount = 5
pooledConnectionsMax = 10

Monitoring
Motivation

- Monitored system and information demands **change over time**
  - Usage measurement and experimentation in software product development
  - Highly dynamic architectures based on microservices
  - Exploration and exploitation with machine learning

- Modeling language determines **possible information types**

- Evolving the modeling language requires a **model re-instantiation**

- Re-instantiations interrupt the **monitoring and phenomena detection processes** and endanger continuous system operation

- A flexible **modeling language** regarding the types of information in the runtime model
  - Makes long living runtime model instances possible and supports continuous adaptive monitoring and system operation
  - Increases the feasibility of runtime models for additional fields of application
Significance of the modeling language

To better understand: Can you show how the modeling language is actually significant?
Information demand changes - Filtering

Query based on types e.g.:
QueryService.
pooledConnectionCount > 5

1.) use
Modeling language implementation with an API to create models

2.) update

3.) access

4.) observe
Monitoring adaptation engine

5.) adapt

Qt: QueryService
<<instanceOf>>
pooledConnectionsCount = 5
pooledConnectionsMax =10

[Brand.2018]
Running system changes - System adaptation

Query based on types e.g.:
QueryService.
pooledConnectionCount > 5

Modeling language implementation with an API to create models

1.) use
2.) instantiate
3.) access

Monitoring
Running system changes - System evolution

1.) re-generate

Modeling language implementation with an API to create models

2.) use

Monitoring

3.) re-instantiate

Discontinuity!

Query based on types e.g.:
QueryService.
pooledConnectionCount > 5

QueryService
:RegionItemFilter
<<instanceOf>>
pooledConnectionsCount: Integer
pooledConnectionsMax: Integer
mode = 51
pooledConnectionsCount = 5
pooledConnectionsMax = 10
<<instanceOf>>
RegionItemFilter
mode: Integer

Meta-model level

Runtime model level

<<instanceOf>>

System representation content

Modeling language definition content
Running system changes - Software evolution

Discontinuity!

Query based on types e.g.: QueryService. pooledConnectionCount > 5
The CompArch approach

There is a way to improve the situation compared to the classical approach!?
Dynamic Object Model pattern

[Diagram showing relationships between ComponentType, Component, PropertyType, Property, and Value classes with multiplicity annotations.]

[Riehle.2005]
The CompArch approach

[Vogel.2018]
Planned roadmap towards a prospective solution

Does this approach fulfill the requirements?

What actually are the important requirements?

How shall the proposed solution be evaluated?

Does this approach fulfill the requirements?
Planned roadmap towards a prospective solution

Survey existing languages

Discover a coherent set of requirements

Describe illustrative scenarios

Running system changes

Information demand changes

Elaborate and evaluate a solution

Validate
Illustrative scenarios and requirements

Can you give us some examples of scenarios and requirements?
Scenarios and requirements overview

Illustrative scenarios

- S1 - System adaptation
- S2 - System evolution
- S3 - Software evolution
- S4 - Systems integration and division
- S5 - Filtering
- S6 - Aggregation
- S7 - Itemization
- S8 - Generalization and specialization

Requirements

- R1 - Updating system representation structure and values
- R2 - Indicating the actual information demand
- R3 - Introducing new classifiers including classifier versions
- R4 - Withdrawing obsolete classifiers
- R5 - Establishing new kinds of relationships
- R6 - Assigning multiple classifiers progressively
- R7 - Integrating multiple classifier systems
- R8 - Introducing new logical elements and relationships
Example system

Simplified mRUBiS runtime model

Multiple tenants
Running system changes
S3 - Software evolution

- Conduct an experiment with new software product version
- Deploy a new version of the QueryService component to early adopter tenants
- Represent new component version with additional properties besides the old

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Classic</th>
<th>ComArch</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3 - Introducing new classifiers including classifier versions</td>
<td>--</td>
<td>(✓)</td>
</tr>
<tr>
<td>S3 - Software evolution</td>
<td>--</td>
<td>(✓)</td>
</tr>
</tbody>
</table>
Information demand changes

S6 - Aggregation - Case 1: Invisible

Aggregation not visible in the runtime model (on the monitoring instrument level)

![Diagram showing the relationship between Runtime model, Monitoring instrument, and Running system, with a note indicating that aggregation is not visible in the runtime model on the monitoring instrument level.]
Information demand changes
S6 - Aggregation - Case 2: Visible

Aggregation visible in the runtime model

Case 2.a: Functional aggregation

Case 2.b: Structural aggregation
Information demand changes

**S6 - Aggregation**

- Represent the service which all query component instances provide together
- Aggregate on the monitoring instrument level
- Provide the sum of exceptions for all early adaptors of query service v2.0.0
- Aggregate on the runtime model level

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Classic</th>
<th>ComArch</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3 - Introducing new classifiers including classifier versions</td>
<td>--</td>
<td>(✓)</td>
</tr>
<tr>
<td>R4 - Withdrawing obsolete classifiers</td>
<td>--</td>
<td>✓</td>
</tr>
<tr>
<td>R5 - Establishing new kinds of relationships</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>R8 - Introducing new logical elements and relationships</td>
<td>--</td>
<td>(✓)</td>
</tr>
<tr>
<td>S6 - Aggregation</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>
Information demand changes

**S8 - Generalization and specialization**

- Indicate potential for configuration optimization by reporting two filters
- Query the number-of-filtered-items property which is common for all filter types
- Consider ten filters of different types in a general way for the query
- Have a specific and a more general classifier assigned to each filter

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Classic</th>
<th>ComArch</th>
</tr>
</thead>
<tbody>
<tr>
<td>R6 - Assigning multiple classifiers progressively</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>S8 - Generalization and specialization</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>
Illustrative scenarios and requirements

What are the remaining scenarios and identified requirements?

How far are the requirements covered by the two approaches you looked at?
## Scenarios and requirements coverage overview

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Requirements</th>
<th>Classical</th>
<th>ComArch</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 - System adaptation</td>
<td>R1 - Updating system representation structure and values</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>S2 - System evolution</td>
<td>R2 - Indicating the actual information demand</td>
<td>(✓)</td>
<td>(✓)</td>
</tr>
<tr>
<td>S3 - Software evolution</td>
<td>R3 - Introducing new classifiers including classifier versions</td>
<td></td>
<td>(✓)</td>
</tr>
<tr>
<td>S4 - Systems integration and division</td>
<td>R4 - Withdrawing obsolete classifiers</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>S5 - Filtering</td>
<td>R5 - Establishing new kinds of relationships</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S6 - Aggregation</td>
<td>R6 - Assigning multiple classifiers progressively</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S7 - Itemization</td>
<td>R7 - Integrating multiple classifier systems</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S8 - Generalization and specialization</td>
<td>R8 - Introducing new logical elements and relationships</td>
<td></td>
<td>(✓)</td>
</tr>
</tbody>
</table>
Summary

- Saw that runtime model modeling languages for flexibility are worth investigating
- Discussed plans on how to elaborate and evaluate a prospective solution
- Discussed the identified requirements

Outlook

- Complete the definition of a coherent set of scenarios and requirements also based on analyzing existing modeling languages
- Elaborate a proposal
- Evaluate regarding cost-effectiveness and support for the requirements
- Consider co-evolution of queries and the runtime model modeling language
References


Thomas Brand and Holger Giese
Hasso Plattner Institute at the University of Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de